C++ VARIABLE TYPES

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive:

There are following basic types of variable in C++ as explained in last chapter:

Type	Description
bool	Stores either value true or false.
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.
wchar_t	A wide character type.

C++ also allows to define various other type of variables which we will cover in subsequent chapters like **Enumeration, Pointer, Array, Reference, Data structures,** and **Classes**.

Following section will cover how to define, declare and use various type of variables.

Variable Declaration in C++:

All variables must be declared before use, although certain declarations can be made implicitly by content. A declaration specifies a type, and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C++ data type including char, w_char, int, float, double, bool or any user defined object etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

A variable declaration with an initializer is always a definition. This means that storage is allocated for the variable and could be declared as follows:

```
int    i = 100;
```

An **extern** declaration is not a definition and does not allocate storage. In effect, it claims that a definition of the variable exists elsewhere in the program. A variable can be declared multiple times in a program, but it must be defined only once. Following is the declaration of a variable with extern keyword:

```
extern int    i;
```

Though you can declare a variable multiple times in your C++ program but it can be decalred only once in a file, a function or a block of code.

Variable Initialization in C++:

Variables are initialized (assigned an value) with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
int d = 3, f = 5;     // initializing d and f.
byte z = 22;          // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x';         // the variable x has the value 'x'.
```

For declarations without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

It is a good programming practice to initialize variables properly otherwise, sometime program would produce unexpected result. Try following example which makes use of various types of variables:

```
#include <iostream>
using namespace std;

int main ()
{
  // Variable declaration:
  int a, b;
  int c;
  float f;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

  cout << c << endl ;

  f = 70.0/3.0;
  cout << f << endl ;

  return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
30
```

```
23.3333
```

Lvalues and Rvalues:

There are two kinds of expressions in C++:

- **lvalue :** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue :** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error:

```
10 = 20;
```